DAUPHINE UNIVERSITÉ PARIS | MIDO MATHÉMATIQUES ET INFORMATIQUE DE LA DÉCISION ET DES ORGANISATIONS

PSL RESEARCH UNIVERSITY

Master M 1

# Projet Java Avancé

**Simulation d'une ECO-résolution en Java: Application au problème des N reines.**

Le problème des N reines consiste à poser sur un damier de N*N cases N reines de façon à ce qu'aucune reine ne soit menacée par une autre reine. Dans ce projet, on cherche une solution distribuée dans laquelle chaque reine est un agent réactif (i.e. un programme autonome qui prend lui-même ses décisions).

Pour corser le problème, on fait l'hypothèse qu'aucun agent ne peut observer directement le damier. Le damier se trouve sur un serveur et chaque agent doit communiquer avec le serveur pour changer sa position. Le serveur accepte le déplacement si la position désirée est libre, sinon, le déplacement est refusé.

Les agents peuvent par contre communiquer entre eux. Dans ce projet, on va supposer que chaque agent va annoncer chacun à son tour dans un ordre préétabli sa position actuelle à tous les autres agents. En recevant ces messages, les agents peuvent mettre à jour leur représentation du damier (qui peut être différente du damier réel) et se déplacer pour minimiser le nombre de reines qui les menacent. Le but du projet est d'étudier si cette solution permet de résoudre le problème des N reines.

L'ECO-résolution (J. Ferber) consiste à répartir une tâche d'optimisation entre plusieurs agents identiques. Chaque agent possède un espace mémoire (ici un damier et une mémoire contenant les $k$ dernières positions annoncées par les agents), un ensemble d'actions (les positions du damier), une action de survie, une fonction économique et un seuil (que la fonction économique ne doit pas franchir). En cas de franchissement du seuil, l'action de survie est déclenchée par l'agent.

On fera ici l'hypothèse que les agents communiquent chacun leur tour selon un ordre préétabli. Les agents communiquent entre eux grâce au protocole UDP. A chaque communication, un agent broadcaste sa position actuelle. Chaque agent conserve les $k$ dernières positions broadcastées. En recevant le message, chaque agent met à jour sa représentation du damier. La fonction économique d'un agent a pour input les positions des reines sur le damier et comme output le nombre de reines qui menacent cet agent. Le seuil est fixé à un : i.e. dès qu'un agent est menacé, il tente de se déplacer sur le damier pour minimiser le nombre des reines qui le menacent. Si le seuil est franchi (i.e. existence d'une ou plusieurs menaces) il déclenche alors son action de survie: elle consiste à oublier la première position reçue dans la liste des messages reçus.

Tous les agents sont connectés à un serveur qui contient le damier de référence et qui est chargé d'affichage le damier. A chacun de ses déplacements, un agent communique avec le serveur pour mettre à jour l'affichage du damier. Les communications avec le serveur se font grâce au protocole TCP. Le serveur doit pouvoir accepter plusieurs connections en même temps. Lors de la communication d'un déplacement, un agent envoie sa position actuelle et la

nouvelle position qu'il désire occuper. Si la position est libre (i.e., il n'y a pas de reine qui occupe cette case), le serveur répond avec un message d'acceptation et effectue le changement sur le damier. Si la position est occupée, le serveur répond à l'aide d'un message négatif et ne modifie pas le damier. La communication se termine. L'agent peut faire une autre demande ultérieurement s'il le souhaite, mais cela implique une nouvelle communication.

On a trouvé une solution lorsqu'aucun agent ne demande de changement de positions et qu'un tour complet d'annonces de positions a été effectué.

**Proposition de modélisation:**

- Classe `Server :`
  Le serveur contient le véritable damier sur lequel sont posées les reines. Au début, le damier est vide. Le serveur communique avec les agents grâce au protocole TCP, il doit pouvoir accepter plusieurs demande en même temps. Lorsqu'un agent se connecte au serveur pour demander un déplacement, l'agent doit fournir la case sur laquelle il se trouve et la case sur laquelle il désire se déplacer. Si celle-ci est libre, le serveur accepte le déplacement en envoyant un message à l'agent et met à jour le damier. Si la case désirée n'est pas libre, le serveur envoie un message à l'agent et ne change pas le damier.

  Le serveur possède aussi une interface graphique qui représente le damier. A chaque déplacement d'une reine, le serveur doit mettre à jour l'interface graphique. On trouvera aussi sur l'interface graphique
  - un bouton «`start`» pour commencer la résolution ;
  - un bouton «`reset`» qui initialise l'état des agents et du damier ;
  - un bouton «`stop`» qui arrête le déplacement des reines. On pourra alors soit continuer la résolution avec le bouton « start » soit réinitialiser l'état des agents et du damier avec le bouton « reset » ;
  - un bouton «`play back`» qui arrête le déplacement des reines et qui rejoue à une vitesse adéquate tous les déplacements des reines ;
  - un bouton «`end`» qui termine la simulation.

  Les actions des boutons supposent une communication entre le serveur et les agents. Vous choisirez vous-même le protocole de communication pour ces fonctionnalités.

- Classe `Agent :` Un agent peut se trouver sur une machine distante du serveur et des autres agents (pour la simulation, agents et serveur seront lancés sur la même machine). Un agent possède une représentation du damier qui est mise à jour avec les annonces de position de chaque agent. Cependant, comme tous les déplacements ne sont pas annoncés et que l'agent ne peut observer le damier du serveur, sa représentation du damier et le vrai damier ne sont pas forcément les mêmes. Pour demander un changement de position, l'agent communique avec le serveur via un protocole TCP. Pour communiquer sa position, l'agent utilise un broadcaste avec le protocole UDP. Agents et serveur doivent aussi communiquer en fonction des boutons de l'interface graphique.

**Implémentation d'une interface graphique en Java.**

Nous utiliserons les composants du package swing (composants à importer en début de programme).
Principe: associer une classe graphique à l'objet devant être visualisé.
Ex. `Manager` `ManagerGUI`
Le constructeur de la classe `ManagerGUI` construit l'interface, la classe propose des méthodes pour interagir avec elle (via l'instance de `ManagerGUI` créée).
Ex.

```
class Manager extends Agent{
  static JFrame Affichage;

  Manager(){
    ManagerGUI mGUI = new ManagerGUI();
    Affichage.setVisible(true);
  }
}
class ManagerGUI implements ActionListener{

  ManagerGUI(int n){
    init();
    JButton bStart = new JButton("Start");
    JButton bReset = new JButton("Reset");
    JButton bStop = new JButton("Stop");
    . . . . . .
    GridLayout d = new GridLayout(n+1,n);
    GridBagConstraints cont = new GridBagConstraints(); //non utilisé
    JFrame fenetre = new JFrame("Projet L3");
    JPanel damier = new JPanel();
    JPanel commande = new JPanel();
    Container c = fenetre.getContentPane();
    c.setLayout(d);
    commande.add(bDame);
    commande.add(bInit);
    c.add(commande);
    bStart.addActionListener(this);
    bReset.addActionListener(this);
    . . . . . .
    damier.setLayout(d);
    . . . . . .
    damier.add(caseDamier, new GridBagConstraints(……));
    . . . . .
    c.add(damier);
    Manager.Affichage =fenetre;
  }
  public void actionPerformed(ActionEvent e){
    String s = e.getActionCommand();
    if (s.equals("Start"))
                System.out.println("Start");
    . . . . .
  }
```

La construction d'une interface fait appel aux classes `JFrame`, `JPanel`, `JButton`, `JTxtField`, `GridLayout`, `GridBagConstraints`. Le principe est basé sur l'utilisation d'un conteneur: un objet fenêtre (instance de `JFrame`) contient des objets graphiques accessibles grâce à la méthode `getContentPane()` de la classe `JFrame`. Cet objet (instance de la classe `Container`) peut être enrichi de composants graphiques avec la méthode `add()`. La disposition des objets dans le containeur est assurée par un gestionnaire de disposition; il en

existe deux types selon qu'il est instance de `FlowLayout` ou `GridLayout`. L'utilisation du second type dispose les objets dans une grille virtuelle dont les dimensions sont déclarées lors de l'instanciation du gestionnaire.
Une des façons d'assurer l'interactivité est de faire correspondre une action aux boutons de l'interface. C'est le rôle du gestionnaire d'action (`Actionlistener`), la classe précédente doit alors implémenter l'interface `ActionListener` est la méthode `actionPerformed(ActionEvent e)` doit être redéfinie. La méthode `addActionListener` de la classe `JButton` permet d'ajouter ce gestionnaire d'action. L'importation du package `java.awt.*` est nécessaire.

**Ce qui est demandé:**

Ecrire un programme Java qui réalise cette simulation:

- Interface graphique avec boutons correspondant aux actions Initialisation, Pas à Pas, Tour Complet, Jeu complet.
- Test sur damier de dimension 4, 6, 8.
- Commentaires et programme-source intégral.